Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №4

Технології паралельних обчислень

Тема: Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірила:
Стеценко І.В.

Київ 2024

ЗМІСТ

1 ЗАВДАННЯ

1. Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework. 20 балів. Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально. 10 балів.
2. Реалізуйте один з алгоритмів комп’ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework. 20 балів.
3. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework. 20 балів.
4. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework. 30 балів.

2 ВИКОНАННЯ

2.1 Аналіз текстів. Бібліотека Apache OpenNLP

Оскільки я цікавлюся машинним навчанням та аналізом текстів, то виконував перше та третє завдання комп’ютерного практикуму за допомогою бібліотеки Apache OpenNLP.

Apache OpenNLP спеціалізується на NLP — Natural Language Processing, що дуже допоможе токенізувати неструктурований природний текст, який містить розділові знаки, скорочення, абревіатури тощо.

Тому імпортуємо у Maven відповідні бібліотеки.

```
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-tools</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-dl</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-uima</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-morfologik-addon</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.opennlp</groupId>
    <artifactId>opennlp-brat-annotator</artifactId>
    <version>2.2.0</version>
</dependency>
```

Для роботи з бібліотекою необхідно завантажити моделі лематизації, розділення речень, токенізації та визначення частин мови.

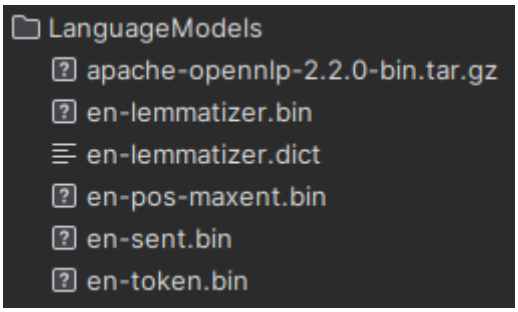


Рисунок 2.1.1 - Лінгвістичні моделі

Розшифрую значення слова “лематизація” — по-простому це перетворення певного слова до його початкової форми, тобто: “слова” — “слово”, “зроблю” — “зробити”, “гарна” — “гарний” тощо.

За допомогою лематизації можна знаходити схожі слова, що потім використаю у третьому завданні.

2.2 Логування. Бібліотека log4j2

Зазвичай для кращої обробки результатів, процесу дебагу програми застосовують логування. У Java є досить популярна бібліотека log4j. Імпортуємо її.

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>2.20.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-jul</artifactId>
    <version>2.20.0</version>
</dependency>
```

Для кожного завдання існують логи, де можна побачити результати ефективності алгоритмів.

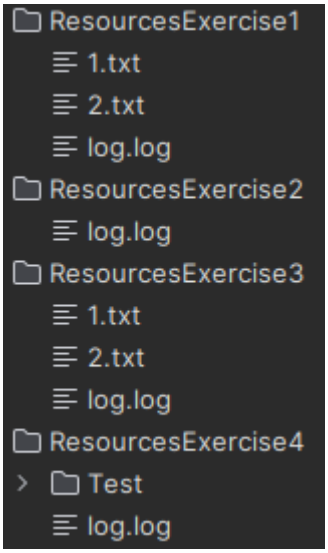


Рисунок 2.2.1 - Логи та допоміжні файли для кожного із завдань

2.3 Перше завдання

Розглянемо структуру проєкту.

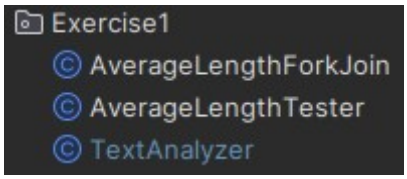


Рисунок 2.3.1 - Структура проєкту

TextAnalyzer — клас, що відповідає за зчитування файлів, розбиття їх на речення та токенізацію речень.

AverageLengthForkJoin — дочірній клас RecursiveTask, що перевизначає метод compute а повертає пару кількості токенів та їхньої суцільної довжини.

AverageLengthTester — клас, який перевіряє роботу однопотокової та багатопотокової версій алгоритму.

Поглянемо на результати ефективності, що знаходяться у ResourcesExercise1/log.log .

```
[FILE: Exercise1_1.txt][SINGLE DURATION: 17][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN
EFFICIENCY: 2.4285714285714284]
[FILE: Exercise1_2.txt][SINGLE DURATION: 17][SINGLE AVERAGE: 4.948142957252978]
```

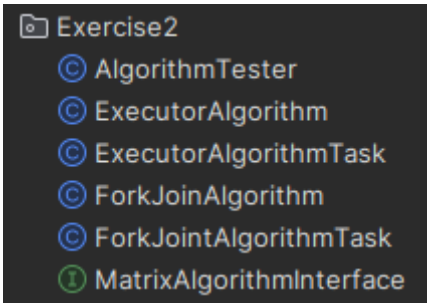
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 34][FORK JOIN EFFICIENCY: 0.5]
[FILE: Exercise1_1.txt][SINGLE DURATION: 15][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN EFFICIENCY: 2.142857142857143]
[FILE: Exercise1_2.txt][SINGLE DURATION: 14][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 14][FORK JOIN EFFICIENCY: 1.0]
[FILE: Exercise1_1.txt][SINGLE DURATION: 51][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN EFFICIENCY: 7.285714285714286]
[FILE: Exercise1_2.txt][SINGLE DURATION: 14][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 13][FORK JOIN EFFICIENCY: 1.0769230769230769]
[FILE: Exercise1_1.txt][SINGLE DURATION: 16][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN EFFICIENCY: 2.2857142857142856]
[FILE: Exercise1_2.txt][SINGLE DURATION: 13][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 11][FORK JOIN EFFICIENCY: 1.1818181818181819]
[FILE: Exercise1_1.txt][SINGLE DURATION: 15][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN EFFICIENCY: 2.142857142857143]
[FILE: Exercise1_2.txt][SINGLE DURATION: 13][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 12][FORK JOIN EFFICIENCY: 1.0833333333333333]
[FILE: Exercise1_1.txt][SINGLE DURATION: 18][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 6][FORK JOIN EFFICIENCY: 3.0]
[FILE: Exercise1_2.txt][SINGLE DURATION: 17][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 15][FORK JOIN EFFICIENCY: 1.1333333333333333]
[FILE: Exercise1_1.txt][SINGLE DURATION: 17][SINGLE AVERAGE: 4.080213903743315]
[FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 7][FORK JOIN EFFICIENCY: 2.4285714285714284]
[FILE: Exercise1_2.txt][SINGLE DURATION: 13][SINGLE AVERAGE: 4.948142957252978]
[FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 12][FORK JOIN EFFICIENCY: 1.0833333333333333]
[FILE: ResourcesExercise1/1.txt][SINGLE DURATION: 20][SINGLE AVERAGE: 4.080213903743315][FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 8]
[FORK JOIN EFFICIENCY: 2.5]
[FILE: ResourcesExercise1/2.txt][SINGLE DURATION: 16][SINGLE AVERAGE: 4.948142957252978][FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 15]
[FORK JOIN EFFICIENCY: 1.0666666666666667]
[FILE: ResourcesExercise1/1.txt][SINGLE DURATION: 20][SINGLE AVERAGE:

```
4.080213903743315][FORK JOIN AVERAGE: 4.080213903743315][FORK JOIN DURATION: 8]
[FORK JOIN EFFICIENCY: 2.5]
[FILE: ResourcesExercise1/2.txt][SINGLE DURATION: 17][SINGLE AVERAGE:
4.948142957252978][FORK JOIN AVERAGE: 4.948142957252978][FORK JOIN DURATION: 16]
[FORK JOIN EFFICIENCY: 1.0625]
```

Як бачимо, чим більший розмір файлу, тим ближче ефективність наближається до 1. Це пов'язано з тим, що моделі важкі та забирають сильно оперативну пам'ять, тому існує лише один екземпляр моделі, який синхронізовано використовують декілька потоків, що знижує швидкість алгоритму.

2.4 Друге завдання

Розглянемо структуру проєкту.



ExecutorAlgorithm та ForkJoinAlgorithm - стрічковий алгоритм множення матриць за допомогою як ForkJoinFramework так і Executor.

ExecutorAlgorithmTask та ForkJoinAlgorithmTask — дочірні класи Runnable та RecursiveAction відповідно, що розбивають обрахунок алгоритмів на частини.

AlgorithmTester — клас для тестування ефективності алгоритмів.

```
[SIZE: 100][SINGLE DURATION: 160][THREADS: 2][EXECUTOR DURATION: 117][EXECUTOR
EFFICIENCY: 1.3675213675213675]
[SIZE: 100][SINGLE DURATION: 160][THREADS: 3][EXECUTOR DURATION: 74][EXECUTOR
EFFICIENCY: 2.1621621621621623]
[SIZE: 100][SINGLE DURATION: 160][THREADS: 4][EXECUTOR DURATION: 60][EXECUTOR
EFFICIENCY: 2.6666666666666665]
[SIZE: 200][SINGLE DURATION: 491][THREADS: 2][EXECUTOR DURATION: 293][EXECUTOR
EFFICIENCY: 1.6757679180887373]
[SIZE: 200][SINGLE DURATION: 491][THREADS: 3][EXECUTOR DURATION: 287][EXECUTOR
```

```
EFFICIENCY: 1.710801393728223]
[SIZE: 200][SINGLE DURATION: 491][THREADS: 4][EXECUTOR DURATION: 280][EXECUTOR
EFFICIENCY: 1.7535714285714286]
[SIZE: 300][SINGLE DURATION: 1296][THREADS: 2][EXECUTOR DURATION: 915][EXECUTOR
EFFICIENCY: 1.4163934426229507]
[SIZE: 300][SINGLE DURATION: 1296][THREADS: 3][EXECUTOR DURATION: 911][EXECUTOR
EFFICIENCY: 1.4226125137211856]
[SIZE: 300][SINGLE DURATION: 1296][THREADS: 4][EXECUTOR DURATION: 959][EXECUTOR
EFFICIENCY: 1.35140771637122]
[SIZE: 400][SINGLE DURATION: 3168][THREADS: 2][EXECUTOR DURATION: 2666][EXECUTOR
EFFICIENCY: 1.1882970742685672]
[SIZE: 400][SINGLE DURATION: 3168][THREADS: 3][EXECUTOR DURATION: 3049][EXECUTOR
EFFICIENCY: 1.0390291898983273]
[SIZE: 400][SINGLE DURATION: 3168][THREADS: 4][EXECUTOR DURATION: 3289][EXECUTOR
EFFICIENCY: 0.9632107023411371]
[SIZE: 200][SINGLE DURATION: 500][FORK JOIN DURATION: 499][FORK JOIN EFFICIENCY:
1.002004008016032]
[SIZE: 300][SINGLE DURATION: 1976][FORK JOIN DURATION: 1726][FORK JOIN
EFFICIENCY: 1.1448435689455387]
[SIZE: 400][SINGLE DURATION: 3662][FORK JOIN DURATION: 2487][FORK JOIN
EFFICIENCY: 1.4724567752312023]
```

2.5 Третє завдання

Розглянемо структуру проєкту.

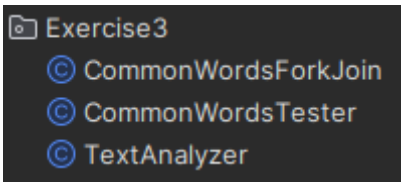


Рисунок 2.5.1 - Структура проєкту

TextAnalyzer - клас, що відповідає за зчитування файлів, розбиття їх на речення, токенізацію, визначення частин мов та лематизацію.

CommonWordsTester — клас, що відповідає за тестування алгоритму.

CommonWordsForkJoin — клас, що відповідає за знаходження спільних слів у документах за допомогою ForkJoinFramework.

```
[SINGLE DURATION: 337][FORK JOIN DURATION: 194][FORK JOIN EFFICIENCY:
1.7371134020618557]
[SINGLE DURATION: 372][FORK JOIN DURATION: 205][FORK JOIN EFFICIENCY:
1.8146341463414635]
```

```
[SINGLE DURATION: 305][FORK JOIN DURATION: 170][FORK JOIN EFFICIENCY:
1.7941176470588236]
[SINGLE DURATION: 279][FORK JOIN DURATION: 179][FORK JOIN EFFICIENCY:
1.558659217877095]
[COMMON WORDS: the lime juice oil salt and cumin in a large bowl add to combine
concern not of will two new serve with world have be for his at marinade or food
highly puree roman pick from list you learn by do it this national recent its
their make year datum practice on which that use offer wish they report profit
after than think sugar well show over one million hand work form analyst say
drink sale europe but building become these only end life time if core under
current speed graphic design any much while he software up good into ]
```

2.6 Четверте завдання

Розглянемо структуру.

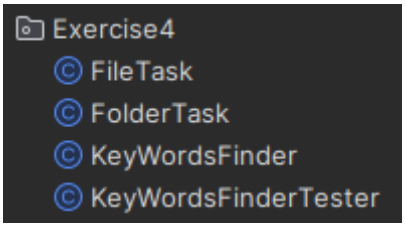


Рисунок 2.6.1 - Структура проєкту

FolderTask — клас, що є підзадачею пошуку директорій всередині директорії.

FileTask — клас, що є підзадачею пошуку ключових слів у файлі.

KeyWordsFinder — алгоритм пошуку ключових слів у файлі. Має версії як однопотокового рекурсивного пошуку так і багатопотокового з ForkJoinFrameWork.

KeyWordsFinderTester — клас, що перевіряє роботу алгоритмів та їхню ефективність.

```
[SINGLE DURATION: 712][FORK JOIN DURATION: 144][FORK JOIN EFFICIENCY:
4.944444444444445]
[SINGLE DURATION: 705][FORK JOIN DURATION: 149][FORK JOIN EFFICIENCY:
4.731543624161074]
[SINGLE DURATION: 791][FORK JOIN DURATION: 145][FORK JOIN EFFICIENCY:
5.455172413793103]
[SINGLE DURATION: 750][FORK JOIN DURATION: 158][FORK JOIN EFFICIENCY:
4.746835443037975]
```

3 ВИСНОВОК

Під час лабораторної роботи опрацювали розробку паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework.

Для вирішення першого та третього завдань була використана бібльотека обробки природної мови Apache OpenNLP, що проводила токенізацію речень, а за допомогою ForkJoinFramework токени оброблювалися у багатопотоковому середовищі. Результати до першого завдання в ефективності алгоритму не є показовими, оскільки ледь перевищують 1. Це можна пояснити тим, що токенізація доволі складний процес, і вона проводилася в одному потоці, оскільки модель є важкою і завантажує мій комп’ютер. Щодо третього завдання, то ефективність стало тримається у межах 1.5 до 1.9, що є відчутним приростом у швидкості.

У другому завданні є версії використання з екзек’юторами та ForkJoinFramework.

У четвертому завданні бачимо суттєвий приріст швидкості, що досягає 6 разів у порівнянні з однопотоковою рекурсивною версією.

Отже, для програміста суттєво знати шляхи для реалізацію паралельних програм, оскільки вони можуть суттєво збільшити швидкість виконання операцій.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-113 курсу

Панченка С. В

```
// ./Lab4/Lab4/src/main/java/org/LabMath/Enums/Coords.java

package org.LabMath.Enums;

public enum Coords {
    Y,
    X
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Vectors/GeneralVector.java

package org.LabMath.Vectors;

import org.LabMath.Interfaces.MathVector;

import java.util.Arrays;

public class GeneralVector implements MathVector<GeneralVector> {
    private static final String ERROR_LENGTHS_NOT_EQUAL = "Lengths of
points arguments are not equal";
    private double[] arguments;

    public GeneralVector(int length) {
        setLength(length);
    }

    public GeneralVector(GeneralVector other) {
        setLength(other.getLength());
        set(other);
    }
```

```
public int getLength() {
    if(arguments == null) {
        return 0;
    }
    return arguments.length;
}

public void setLength(int length) {
    var currentLength = getLength();

    if(currentLength==length) return;

    var minLength = Math.min(currentLength, length);
    var args = new double[length];
    for(var i = 0; i < minLength; ++i) {
        args[i] = getAt(i);
    }

    arguments = args;
}

@Override
public void set(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, other.getAt(i));
    }
}

@Override
public GeneralVector clone() {
    return new GeneralVector(this);
```

```
    }

    @Override
    public String toString() {
        return Arrays.toString(arguments);
    }

    private void checkSizesEqual(GeneralVector other) {
                                assert getLength() == other.getLength() :
ERROR_LENGTHS_NOT_EQUAL;
    }

    @Override
    public void add(GeneralVector other) {
        checkSizesEqual(other);
        for(var i = 0; i < getLength(); ++i) {
            setAt(i, getAt(i) + other.getAt(i));
        }
    }

    @Override
    public void add(double value) {
        for(var i = 0; i < getLength(); ++i) {
            setAt(i, getAt(i) + value);
        }
    }

    @Override
    public void sub(GeneralVector other) {
        checkSizesEqual(other);
        for(var i = 0; i < getLength(); ++i) {
            setAt(i, getAt(i) - other.getAt(i));
```

```
    }
}

@Override
public void sub(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) - value);
    }
}

@Override
public void mul(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * other.getAt(i));
    }
}

@Override
public void mul(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * value);
    }
}

@Override
public void div(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / other.getAt(i));
    }
}
```

```
@Override
public void div(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / value);
    }
}

@Override
public double getSize() {
    return Math.sqrt(getSizeSquared());
}

@Override
public double getSizeSquared() {
    double s = 0;
    for(var i = 0; i < getLength(); ++i) {
        s += Math.pow(getAt(i), 2);
    }
    return s;
}

@Override
public double getDotProduct(GeneralVector other) {
    checkSizesEqual(other);
    var prod = 0;
    for(var i = 0; i < getLength(); ++i) {
        prod += getAt(i) * other.getAt(i);
    }
    return prod;
}
```

```
@Override
public double getDistance(GeneralVector other) {
    checkSizesEqual(other);
    var dist = 0.0;
    for(var i = 0; i < getLength(); ++i) {
        dist += Math.pow(getAt(i) - other.getAt(i), 2);
    }
    dist = Math.sqrt(dist);
    return dist;
}

@Override
public GeneralVector getForwardVector() {
    var forwardVec = clone();
    var size = getSize();
    for(var i = 0; i < getLength(); ++i) {
        forwardVec.setAt(i, getAt(i) / size);
    }
    return forwardVec;
}

@Override
public double getAt(int index) {
    return arguments[index];
}

@Override
public void setAt(int index, double value) {
    arguments[index] = value;
}

@Override
```

```
    public GeneralVector getOpposite() {
        var v = clone();
        v.toOpposite();
        return v;
    }

    @Override
    public void toOpposite() {
        for(var i = 0; i < getLength(); ++i) {
            setAt(i, -getAt(i));
        }
    }

}


// ./Lab4/Lab4/src/main/java/org/LabMath/Vectors/Vector2D.java

package org.LabMath.Vectors;

import org.LabMath.Enums.*;
import org.LabMath.Interfaces.MathVector;

public class Vector2D implements MathVector<Vector2D> {
    private final GeneralVector vec = new GeneralVector(2);

    public Vector2D() {}

    public Vector2D(double y, double x) {
        set(y, x);
    }
```

```
public Vector2D(Vector2D other) {
    set(other);
}

public double getX() {
    return getAt(Coords.X.ordinal());
}

public double getY() {
    return getAt(Coords.Y.ordinal());
}

public void set(Vector2D other) {
    vec.set(other.vec);
}

public void set(double y, double x) {
    setX(x);
    setY(y);
}

public void setX(double value) {
    setAt(Coords.X.ordinal(), value);
}

public void setY(double value) {
    setAt(Coords.Y.ordinal(), value);
}

@Override
public Vector2D clone() {
    return new Vector2D(getY(), getX());
```

```
}

@Override
public String toString() {
    return vec.toString();
}

@Override
public void add(Vector2D other) {
    vec.add(other.vec);
}

@Override
public void add(double value) {
    vec.add(value);
}

@Override
public void sub(double value) {
    vec.sub(value);
}

@Override
public void sub(Vector2D other) {
    vec.sub(other.vec);
}

@Override
public void mul(Vector2D other) {
    vec.mul(other.vec);
}
```

```
@Override
public void mul(double value) {
    vec.mul(value);
}

@Override
public void div(Vector2D other) {
    vec.div(other.vec);
}

@Override
public void div(double value) {
    vec.div(value);
}

@Override
public double getSize() {
    return vec.getSize();
}

@Override
public double getSizeSquared() {
    return vec.getSizeSquared();
}

@Override
public double getDotProduct(Vector2D other) {
    return vec.getDotProduct(other.vec);
}

@Override
public double getDistance(Vector2D other) {
```

```
        return vec.getDistance(other.vec);
    }

    @Override
    public Vector2D getForwardVector() {
        var forwardVec = clone();
        forwardVec.vec.set(forwardVec.vec.getForwardVector());
        return forwardVec;
    }

    @Override
    public double getAt(int index) {
        return vec.getAt(index);
    }

    @Override
    public void setAt(int index, double value) {
        vec.setAt(index, value);
    }

    @Override
    public Vector2D getOpposite() {
        var v = clone();
        v.toOpposite();
        return v;
    }

    @Override
    public void toOpposite() {
        vec.toOpposite();
    }
}
```

```
// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/MathVector.java

package org.LabMath.Interfaces;

import org.LabMath.Interfaces.General.*;

public interface MathVector<T> extends Cloneable, Divisible<T>,
Multipliable<T>, Addable<T>, Subtractable<T>,
                        DoubleDivisible, DoubleMultipliable, DoubleAddable,
DoubleSubtractable {
    double getSize();
    double getSizeSquared();
    double getDotProduct(T other);
    double getDistance(T other);
    T getForwardVector();
    double getAt(int index);
    void setAt(int index, double value);
    T getOpposite();
    void toOpposite();
    void set(T other);
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/MathMatrix.java

package org.LabMath.Interfaces;

import org.LabMath.Interfaces.General.*;

public interface MathMatrix<T> extends Cloneable, Addable<T>,
```

```
Subtractable<T>, Divisible<T>,
                        GetMultipliable<T>,  Settable<T>,  DoubleSubtractable,
DoubleMultipliable, DoubleAddable, DoubleDivisible {
    int[] getDimensions();
    double getAt(int... indexes);
    void setAt(double value, int... indexes);
    int calcIndex(int... indexes);
}


//
./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/DoubleDivisible.java

package org.LabMath.Interfaces.General;

public interface DoubleDivisible {
    void div(double other);
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/Divisible.java

package org.LabMath.Interfaces.General;

import jdk.jshell.spi.ExecutionControl;

public interface Divisible<T> {
    void div(T other) throws ExecutionControl.NotImplementedException;
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/Addable.java
```

```
package org.LabMath.Interfaces.General;

public interface Addable<T> {
   void add(T other);
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/Subtractable.java

package org.LabMath.Interfaces.General;

public interface Subtractable<T> {
   void sub(T other);
}


//
./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/DoubleAddable.java

package org.LabMath.Interfaces.General;

public interface DoubleAddable {
   void add(double other);
}


//
./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/DoubleSubtractable.java

package org.LabMath.Interfaces.General;
```

```
public interface DoubleSubtractable {
    void sub(double other);
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/Settable.java

package org.LabMath.Interfaces.General;

public interface Settable<T> {
    void set(T other);
}


//
./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/DoubleMultipliable.java

package org.LabMath.Interfaces.General;

public interface DoubleMultipliable {
    void mul(double other);
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/Multipliable.java

package org.LabMath.Interfaces.General;

public interface Multipliable<T> {
    void mul(T other);
}
```

```
        //
./Lab4/Lab4/src/main/java/org/LabMath/Interfaces/General/GetMultipliable.java

        package org.LabMath.Interfaces.General;

        import jdk.jshell.spi.ExecutionControl;

        public interface GetMultipliable<T> {
          T getMul(T other) throws ExecutionControl.NotImplementedException;
        }


        // ./Lab4/Lab4/src/main/java/org/LabMath/Matrixes/Matrix2DFactory.java

        package org.LabMath.Matrixes;

        public class Matrix2DFactory {

            public Matrix2DFactory() {}

            public static void main(String[] args) {
                var factory = new Matrix2DFactory();
                var minVal = 0;
                var maxVal = 10;
                var rows = 5;
                var cols = 6;
                var one = factory.getRandom(rows, cols, minVal, maxVal);
                var two = factory.getRandom(cols, rows, minVal, maxVal);
                var result = one.getMul(two);
                System.out.println(result);
            }
```

```
    public Matrix2D getRandom(int rows, int cols, int minVal, int maxVal) {
        var res = new Matrix2D(rows, cols);
        for(var i = 0; i < rows; ++i) {
            for(var j = 0; j < cols; ++j) {
                res.setAt(Math.random() * (maxVal - minVal) + minVal, i, j);
            }
        }
        return res;
    }
}


// ./Lab4/Lab4/src/main/java/org/LabMath/Matrixes/GeneralMatrix.java

package org.LabMath.Matrixes;

import org.LabMath.Interfaces.MathMatrix;
import org.LabMath.Vectors.GeneralVector;
import jdk.jshell.spi.ExecutionControl;

import java.util.Arrays;

public final class GeneralMatrix implements MathMatrix<GeneralMatrix> {
    private static final String ERROR_INDEXES = "Indexes are less than 0";
    private static final String ERROR_DIMENSIONS = "Matrix dimensions not
equal";
      private static final String ERROR_DIMENSION_INDEXES = "Indexes
length is not equal to amount of dimension";
    private final int[] dimensions;
    private final int total;
    private final GeneralVector mat;
```

```
public GeneralMatrix(int... dimensions) {
    this.dimensions = dimensions.clone();
    var t = 1;
    for(var d : dimensions) t *= d;
    this.total = t;
    this.mat = new GeneralVector(this.total);
}

private String doDraw(int[] indexes, int dimension) {
    var res = new StringBuilder();
    res.append("{");
    for(var i = 0; i < this.dimensions[dimension]; ++i) {
        indexes[dimension] = i;
        if(dimension == this.dimensions.length - 1) {
            res.append(this.mat.getAt(calcIndex(indexes)));
        } else {
            res.append(doDraw(indexes, dimension + 1));
        }
        res.append(this.dimensions[dimension] - 1 == i ? "" : ", ");
    }
    res.append("}");
    return res.toString();
}

@Override
public String toString() {
    var indexes = new int[this.dimensions.length];
    return doDraw(indexes, 0);
}

private void checkDimensions(int[] dimensions) {
```

```
    if(!Arrays.equals(this.dimensions, dimensions)) {
        throw new IllegalArgumentException(ERROR_DIMENSIONS);
    }
}

private void checkIndexes(int[] indexes) {
    if(!Arrays.stream(indexes).allMatch(e -> e >= 0)) {
        throw new IllegalArgumentException(ERROR_INDEXES);
    }
}

@Override
public void add(GeneralMatrix other) {
    checkDimensions(other.dimensions);
    for(var i = 0; i < total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + other.mat.getAt(i));
    }
}

@Override
public void add(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + value);
    }
}

@Override
public void div(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) / value);
    }
}
```

```
        @Override
        public void mul(double value) {
            for(var i = 0; i < this.total; ++i) {
                this.mat.setAt(i, this.mat.getAt(i) * value);
            }
        }

        @Override
        public void sub(double value) {
            for(var i = 0; i < this.total; ++i) {
                this.mat.setAt(i, this.mat.getAt(i) - value);
            }
        }

        @Override
                public GeneralMatrix getMul(GeneralMatrix other) throws
ExecutionControl.NotImplementedException {
            throw new ExecutionControl.NotImplementedException("");
        }

        @Override
        public void set(GeneralMatrix other) {
            checkDimensions(other.dimensions);
            for(var i = 0; i < this.total; ++i) {
                this.mat.setAt(i, this.mat.getAt(i));
            }
        }

        @Override
        public void sub(GeneralMatrix other) {
            checkDimensions(other.dimensions);
```

```
        for(var i = 0; i < total; ++i) {
            this.mat.setAt(i, other.mat.getAt(i) - other.mat.getAt(i));
        }
    }

    @Override
    public int[] getDimensions() {
        return dimensions.clone();
    }

    @Override
    public double getAt(int... indexes) {
        checkIndexes(indexes);
        return this.mat.getAt(this.calcIndex(indexes));
    }

    @Override
    public void setAt(double value, int... indexes) {
        checkIndexes(indexes);
        var index = this.calcIndex(indexes);
        this.mat.setAt(index, value);
    }

    @Override
                    public void div(GeneralMatrix other) throws
ExecutionControl.NotImplementedException {
        throw new ExecutionControl.NotImplementedException("");
    }

    @Override
    public int calcIndex(int... indexes) {
        if(indexes.length != dimensions.length) {
```

```
                throw new IllegalArgumentException(ERROR_DIMENSION_INDEXES);
            }
            var index = 0;
            var mult = 1;
            for(var i : dimensions) mult *= i;
            for(var i = 0; i < indexes.length; ++i) {
                mult /= dimensions[i];
                index += indexes[i] * mult;
            }
            return index;
        }
    }


    // ./Lab4/Lab4/src/main/java/org/LabMath/Matrixes/Matrix2D.java

    package org.LabMath.Matrixes;

    import org.LabMath.Interfaces.MathMatrix;
    import jdk.jshell.spi.ExecutionControl;

    public class Matrix2D implements MathMatrix<Matrix2D> {
        private static final String ERROR_MULTIPLICATION = "Rows and columns are not equal";
        private static final String ERROR_INDEXES = "Indexes are less than 0";
        private final int rows;
        private final int cols;
        private final GeneralMatrix mat;

        public static void main(String[] args) {}
```

```
        @Override
        public String toString() {
            return mat.toString();
        }

        public Matrix2D(int rows, int cols) {
            mat = new GeneralMatrix(rows, cols);
            this.rows = rows;
            this.cols = cols;
        }

        public int getRows() {
            return this.rows;
        }

        public int getCols() {
            return this.cols;
        }

        @Override
        public void add(Matrix2D other) {
            this.mat.add(other.mat);
        }

        @Override
                        public void div(Matrix2D other) throws
ExecutionControl.NotImplementedException {
            throw new ExecutionControl.NotImplementedException("");
        }

        @Override
        public void add(double value) {
```

```
    this.mat.add(value);
}

@Override
public void div(double value) {
    this.mat.div(value);
}

@Override
public void mul(double value) {
    this.mat.mul(value);
}

@Override
public void sub(double value) {
    this.mat.sub(value);
}

@Override
public Matrix2D getMul(Matrix2D other) {
    var cols = getCols();

    assert cols == other.getRows() : ERROR_MULTIPLICATION;

    var result = new Matrix2D(rows, cols);

    for(var i = 0; i < getRows(); ++i) {
        for(var j = 0; j < other.getCols(); ++j) {
            var value = 0;
            for(var k = 0; k < cols; ++k) {
                value += this.mat.getAt(i, k) * other.mat.getAt(k, j);
            }
```

```
            result.setAt(value, i, j);
        }
    }

    return result;
}

@Override
public void set(Matrix2D other) {
    this.mat.set(other.mat);
}

@Override
public void sub(Matrix2D other) {
    this.mat.sub(other.mat);
}

@Override
public int[] getDimensions() {
    return this.mat.getDimensions();
}

@Override
public double getAt(int... indexes) {
    if(indexes.length != 2) {
        throw new IllegalArgumentException(ERROR_INDEXES);
    }
    return this.mat.getAt(indexes);
}

@Override
public void setAt(double value, int... indexes) {
```

```
            if(indexes.length != 2) {
                throw new IllegalArgumentException(ERROR_INDEXES);
            }
            this.mat.setAt(value, indexes);
        }

        @Override
        public int calcIndex(int... indexes) {
            return this.mat.calcIndex(indexes);
        }

        public boolean isSquare() {
            return this.rows == this.cols;
        }
    }


    //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise3/CommonWordsTester.java

    package org.LabExercises.Exercise3;

    import org.apache.logging.log4j.LogManager;
    import org.apache.logging.log4j.Logger;

    import java.util.Arrays;

    public class CommonWordsTester {
        private static final Logger LOGGER =
LogManager.getLogger("Exercise3");
        private static final String FILE_ONE = "ResourcesExercise3/1.txt";
        private static final String FILE_TWO = "ResourcesExercise3/2.txt";
```

```
    public static void main(String[] args) {
        testProfile();
        testCorrectness();
    }

    private static void testProfile() {
        var textAnalyzer = TextAnalyzer.getInstance();

        var builder = new StringBuilder();

        var singleStartTime = System.currentTimeMillis();
        textAnalyzer.getCommonWords(FILE_ONE, FILE_TWO);
        var singleDuration = System.currentTimeMillis() - singleStartTime;

        builder.append("[SINGLE DURATION: ").append(singleDuration).append("]");

        var startTime = System.currentTimeMillis();
        textAnalyzer.getCommonWordsForkJoin(FILE_ONE, FILE_TWO);
        var duration = System.currentTimeMillis() - startTime;

        builder.append("[FORK JOIN DURATION: ").append(duration).append("]");
        builder.append("[FORK JOIN EFFICIENCY: ").append((double) singleDuration / duration).append("]");

        LOGGER.info(builder.toString());
    }

    private static void testCorrectness() {
```

```
                var textAnalyzer = TextAnalyzer.getInstance();
                var res = textAnalyzer.getCommonWords(FILE_ONE,
FILE_TWO);
                var builder = new StringBuilder();
                builder.append("[COMMON WORDS: ");
                Arrays.stream(res).forEach(s -> builder.append(s).append(" "));
                builder.append("]");
                LOGGER.info(builder.toString());
            }
        }


        //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise3/CommonWordsForkJoin.java

        package org.LabExercises.Exercise3;

        import java.util.concurrent.RecursiveAction;
        import java.util.List;

        public class CommonWordsForkJoin extends RecursiveAction {
            private final List<String> commonWords;
            private final List<String> first;
            private final List<String> second;
            private final int index;

            public CommonWordsForkJoin(int index, List<String> first, List<String>
second, List<String> commonWords) {
                this.commonWords = commonWords;
                this.first = first;
                this.second = second;
                this.index = index;
```

```
        }

        @Override
        protected void compute() {
            if(index >= first.size()) return;

            var rhs = new CommonWordsForkJoin(index + 1, first, second,
commonWords);
            rhs.fork();

            if(second.contains(first.get(index))) {
                synchronized(commonWords) {
                    commonWords.add(first.get(index));
                }
            }

            rhs.join();
        }
    }


    // ./Lab4/Lab4/src/main/java/org/LabExercises/Exercise3/TextAnalyzer.java

    package org.LabExercises.Exercise3;

    import opennlp.tools.lemmatizer.LemmatizerME;
    import opennlp.tools.lemmatizer.LemmatizerModel;
    import opennlp.tools.postag.POSModel;
    import opennlp.tools.postag.POSTaggerME;
    import opennlp.tools.sentdetect.SentenceDetectorME;
    import opennlp.tools.sentdetect.SentenceModel;
    import opennlp.tools.tokenize.TokenizerME;
```

```
import opennlp.tools.tokenize.TokenizerModel;
import org.LabExercises.Exercise1.AverageLengthForkJoin;
import org.springframework.data.util.StreamUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.ForkJoinPool;
import java.util.stream.Stream;

public class TextAnalyzer {
    private static final String SENTENCE_MODEL_PATH =
"LanguageModels/en-sent.bin";
    private static final String TOKEN_MODEL_PATH =
"LanguageModels/en-token.bin";
    private static final String LEMMATIZER_MODEL_PATH =
"LanguageModels/en-lemmatizer.bin";
    private static final String POSTAGGER_MODEL_PATH =
"LanguageModels/en-pos-maxent.bin";
    private static final String NOT_VALID_SYMBOLS = "[^a-zA-Z\s\n]";
    private static final String WHITE_SPACE = " ";
    private static final String LINE_FEED = "\n";
    private static final String EMPTY_SPACE = "";
    private final ForkJoinPool pool = ForkJoinPool.commonPool();
    private final SentenceDetectorME sentenceDetector;
    private final TokenizerME tokenizer;
    private final LemmatizerME lemmatizer;
    private final POSTaggerME posTagger;
```

```
        private static TextAnalyzer fileStringReader;

        public static TextAnalyzer getInstance() {
            if (fileStringReader == null) {
                fileStringReader = new TextAnalyzer();
            }
            return fileStringReader;
        }

        private TextAnalyzer() {
            try {
                var sentModelIn = new FileInputStream(SENTENCE_MODEL_PATH);
                var sentModel = new SentenceModel(sentModelIn);
                sentenceDetector = new SentenceDetectorME(sentModel);
                var tokModelIn = new FileInputStream(TOKEN_MODEL_PATH);
                var tokModel = new TokenizerModel(tokModelIn);
                tokenizer = new TokenizerME(tokModel);
                var lemModelIn = new FileInputStream(LEMMATIZER_MODEL_PATH);
                var lemModel = new LemmatizerModel(lemModelIn);
                lemmatizer = new LemmatizerME(lemModel);
                var posModelIn = new FileInputStream(POSTAGGER_MODEL_PATH);
                var posModel = new POSModel(posModelIn);
                posTagger = new POSTaggerME(posModel);
            } catch(IOException e) {
                throw new RuntimeException(e);
            }
        }
```

```
        public String[] getCommonWords(String fileNameOne, String
fileNameTwo) {
                var tokens = Stream.of(fileNameOne, fileNameTwo)
                        .map(s -> getTokenStream(s).toList()).toList();

                var tagsStream = tokens.stream().map(
                        s -> s.stream().map(posTagger::tag).toList());

                var lemmas = StreamUtils.zip(tokens.stream(), tagsStream, (tokMat,
tagMat) ->
                                StreamUtils.zip(tokMat.stream(),          tagMat.stream(),
lemmatizer::lemmatize)
                                        .flatMap(Arrays::stream).distinct().toList()).toList();

                return lemmas.get(0).stream().filter(lemmas.get(1)::contains)
                        .toArray(String[]::new);
        }

        public String[] getCommonWordsForkJoin(String fileNameOne, String
fileNameTwo) {
                var tokens = Stream.of(fileNameOne, fileNameTwo)
                        .map(s -> getTokenStream(s).toList()).toList();

                var tagsStream = tokens.stream().map(
                        s -> s.stream().map(posTagger::tag).toList());

                var lemmas = StreamUtils.zip(tokens.stream(), tagsStream, (tokMat,
tagMat) ->
                                StreamUtils.zip(tokMat.stream(),          tagMat.stream(),
lemmatizer::lemmatize)
                                        .flatMap(Arrays::stream).distinct().toList()).toList();
```

```
        var commonWords = new ArrayList<String>();
        pool.invoke(new CommonWordsForkJoin(0, lemmas.get(0),
lemmas.get(1), commonWords));

        return commonWords.toArray(String[]::new);
    }

    public Stream<String[]> getTokenStream(String fileName) {
        try {
            String text = readFile(fileName);
            return Arrays.stream(sentenceDetector.sentDetect(text))
                .map(s -> s.replaceAll(NOT_VALID_SYMBOLS,
EMPTY_SPACE))
                .map(tokenizer::tokenize);
        } catch(FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    public double getAverageLength(String filePath) {
        var average = getTokenStream(filePath).flatMap(Arrays::stream)
            .mapToDouble(String::length).average();
        return average.isPresent() ? average.getAsDouble() : 0;
    }

    public double getAverageLengthForkJoin(String filePath) {
        var tokens = getTokenStream(filePath).toArray(String[][]::new);
        var res = pool.invoke(new AverageLengthForkJoin(0, tokens));
        return (double) res.getFirst() / res.getSecond();
    }
```

```
        public String readFile(String filePath) throws FileNotFoundException {
            var file = new File(filePath);
            var scanner = new Scanner(file);
            var builder = new StringBuilder();
            while(scanner.hasNextLine()) {
                builder.append(scanner.nextLine()).append(LINE_FEED);
            }
            return builder.toString();
        }
    }
```

```
    //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/MatrixAlgorithmInterface.jav
a

    package org.LabExercises.Exercise2;

    import org.LabMath.Matrixes.Matrix2D;

    public interface MatrixAlgorithmInterface {
        Matrix2D solve();
    }
```

```
    //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/ExecutorAlgorithm.java

    package org.LabExercises.Exercise2;

    import org.LabMath.Matrixes.Matrix2D;
    import org.LabMath.Matrixes.Matrix2DFactory;
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.function.IntSupplier;

public class ExecutorAlgorithm implements MatrixAlgorithmInterface {
    private static final String ERROR_MULTIPLICATION = "Rows and columns are not equal";
    private final Matrix2D first;
    private final Matrix2D second;
    private final ThreadPoolExecutor executor;

    public ExecutorAlgorithm(int threadsNum, Matrix2D first, Matrix2D second) {
        this.first = first;
        this.second = second;
        executor = (ThreadPoolExecutor) Executors.newFixedThreadPool(threadsNum);
    }

    @Override
    public Matrix2D solve() {
        if(first.getCols() != second.getRows()) {
            throw new IllegalArgumentException(ERROR_MULTIPLICATION);
        }

        var firstRows = first.getRows();
        var poolSize = executor.getMaximumPoolSize();
```

```
                var result = new Matrix2D(firstRows, second.getCols());
                var isRowsLess = firstRows < poolSize;
                var totalThreads = isRowsLess ? firstRows : poolSize;
                var step = isRowsLess ? 1 : poolSize;
                var endController = new CountDownLatch(totalThreads);

                for(var i = 0; i < totalThreads; ++i) {
                        var task = new ExecutorAlgorithmTask(step, i, first,
                                second, result, endController);
                        executor.execute(task);
                }

                try {
                        endController.await();
                } catch(InterruptedException ex) {
                        ex.printStackTrace();
                }

                executor.shutdown();

                return result;
        }

}


//
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/ExecutorAlgorithmTask.java

package org.LabExercises.Exercise2;

import org.LabMath.Matrixes.Matrix2D;
```

```
import java.util.concurrent.CountDownLatch;

public class ExecutorAlgorithmTask implements Runnable {
    private final Matrix2D first;
    private final Matrix2D second;
    private final Matrix2D result;
    private final int step;
    private final int firstRow;
    private final CountDownLatch endController;

    public ExecutorAlgorithmTask(int step, int firstRow, Matrix2D first,
Matrix2D second,
                                 Matrix2D result,
CountDownLatch endController) {
        this.first = first;
        this.second = second;
        this.result = result;
        this.step = step;
        this.firstRow = firstRow;
        this.endController = endController;
    }

    @Override
    public void run() {
        var firstRows = first.getRows();
        var firstCols = first.getCols();
        var secondCols = second.getCols();
        var curRow = firstRow;
        while(curRow < firstRows) {
            for(var j = 0; j < secondCols; ++j) {
                var value = 0.0;
```

```
                        for(var k = 0; k < firstCols; ++k) {
                            var lhs = first.getAt(curRow, k);
                            var rhs = second.getAt(k, j);
                            value +=  lhs * rhs;
                        }
                        result.setAt(value, curRow, j);
                    }
                    curRow += step;
                }
                endController.countDown();
            }
        }

        //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/ForkJoinAlgorithm.java

        package org.LabExercises.Exercise2;

        import org.LabMath.Matrixes.Matrix2D;
        import org.LabMath.Matrixes.Matrix2DFactory;

        import java.io.FileWriter;
        import java.io.IOException;
        import java.util.concurrent.CountDownLatch;
        import java.util.concurrent.ForkJoinPool;

        public class ForkJoinAlgorithm implements MatrixAlgorithmInterface {
            private static final String ERROR_MULTIPLICATION = "Rows and
columns are not equal";
            private final ForkJoinPool pool = ForkJoinPool.commonPool();
            private final Matrix2D first;
            private final Matrix2D second;
```

```
        public ForkJoinAlgorithm(Matrix2D first, Matrix2D second) {
            this.first = first;
            this.second = second;
        }

        @Override
        public Matrix2D solve() {
            if(first.getCols() != second.getRows()) {
                throw new IllegalArgumentException(ERROR_MULTIPLICATION);
            }

            var result = new Matrix2D(first.getRows(), second.getCols());
            var task = new ForkJointAlgorithmTask(0, first, second, result);
            pool.invoke(task);

            return result;

        }
    }


    // ./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/AlgorithmTester.java

    package org.LabExercises.Exercise2;

    import org.LabExercises.Exercise2.ExecutorAlgorithm;
    import org.LabExercises.Exercise2.ForkJoinAlgorithm;
    import org.LabMath.Matrixes.Matrix2D;
    import org.LabMath.Matrixes.Matrix2DFactory;
    import org.apache.logging.log4j.LogManager;
```

```
import org.apache.logging.log4j.Logger;

import java.util.stream.IntStream;

public class AlgorithmTester {
    private static final Matrix2DFactory MATRIX_2_D_FACTORY = new Matrix2DFactory();
    private static final Logger LOGGER = LogManager.getLogger("Exercise2");
    private static final int MIN_VAL = 0;
    private static final int MAX_VAL = 10;

    private static Matrix2D getTestMatrix(int size) {
        return MATRIX_2_D_FACTORY.getRandom(size, size, MIN_VAL, MAX_VAL);
    }

    public static void main(String[] args) {
        testExecutor();
        testForkJoin();
    }

    private static void testForkJoin() {
        IntStream.of(100, 200, 300, 400).forEach(s -> {
            var mat = getTestMatrix(s);
            var startTime = System.currentTimeMillis();
            mat.getMul(mat);
            var singleDuration = System.currentTimeMillis() - startTime;

            var start = System.currentTimeMillis();
            var forkJoinAlg = new ForkJoinAlgorithm(mat, mat);
            forkJoinAlg.solve();
```

```
                var d = System.currentTimeMillis() - start;

                var builder = new StringBuilder();
                builder.append("[SIZE: ").append(s).append("]");
                builder.append("[SINGLE DURATION: ").append(singleDuration).append("]");
                builder.append("[FORK JOIN DURATION: ").append(d).append("]");
                builder.append("[FORK JOIN EFFICIENCY: ").append((double) singleDuration / d).append("]");
                LOGGER.info(builder.toString());
            });
        }

        private static void testExecutor() {
            var builder = new StringBuilder();
            IntStream.of(100, 200, 300, 400).forEach(s -> {
                var mat = getTestMatrix(s);
                var startTime = System.currentTimeMillis();
                mat.getMul(mat);
                var singleDuration = System.currentTimeMillis() - startTime;
                IntStream.of(2, 3, 4).forEach(n -> {
                    builder.append("[SIZE: ").append(s).append("]");
                    builder.append("[SINGLE DURATION: ").append(singleDuration).append("]");
                    builder.append("[THREADS: ").append(n).append("]");

                    var start = System.currentTimeMillis();
                    var executorAlg = new ExecutorAlgorithm(n, mat, mat);
                    executorAlg.solve();
                    var d = System.currentTimeMillis() - start;
```

```
                builder.append("[EXECUTOR DURATION:
").append(d).append("]");
                builder.append("[EXECUTOR EFFICIENCY:
").append((double) singleDuration / d).append("]");
                LOGGER.info(builder.toString());
                builder.setLength(0);
            });
        });
    }
}


//
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise2/ForkJointAlgorithmTask.java

package org.LabExercises.Exercise2;

import org.LabMath.Matrixes.Matrix2D;

import java.util.concurrent.RecursiveAction;

public class ForkJointAlgorithmTask extends RecursiveAction {
    private final Matrix2D first;
    private final Matrix2D second;
    private final Matrix2D result;
    private final int row;

    public ForkJointAlgorithmTask(int row, Matrix2D first,
                                  Matrix2D second, Matrix2D
result) {
        this.first = first;
        this.second = second;
```

```
            this.result = result;
            this.row = row;
        }

        @Override
        public void compute() {
            if(row >= result.getRows()) {
                return;
            }

            var rhsTask = new ForkJointAlgorithmTask(row + 1, first, second,
result);
            rhsTask.fork();

            var firstCols = first.getCols();

            for(var j = 0; j < second.getCols(); ++j) {
                var val = 0.0;
                for(var k = 0; k < firstCols; ++k) {
                    val += first.getAt(row, k) * second.getAt(k, j);
                }
                result.setAt(val, row, j);
            }

            rhsTask.join();
        }
    }


    //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise1/AverageLengthTester.java
```

```
package org.LabExercises.Exercise1;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.IOException;

public class AverageLengthTester {
    private static final Logger LOGGER = LogManager.getLogger("Exercise1");
    private static final String FILE_ONE = "ResourcesExercise1/1.txt";
    private static final String FILE_TWO = "ResourcesExercise1/2.txt";

    public static void main(String[] args) throws IOException {
        testProfile(FILE_ONE);
        testProfile(FILE_TWO);
    }

    private static void testProfile(String fileName) throws IOException {
        var textAnalyzer = TextAnalyzer.getInstance();

        var builder = new StringBuilder();
        builder.append("[FILE: ").append(fileName).append("]");

        var singleStartTime = System.currentTimeMillis();
        var singleAverage = textAnalyzer.getAverageLength(fileName);
        var singleDuration = System.currentTimeMillis() - singleStartTime;

        builder.append("[SINGLE DURATION: ").append(singleDuration).append("]");
        builder.append("[SINGLE AVERAGE: ").append(singleAverage).append("]");
```

```
                var startTime = System.currentTimeMillis();
                var average = textAnalyzer.getAverageLengthForkJoin(fileName);
                var duration = System.currentTimeMillis() - startTime;

                builder.append("[FORK JOIN AVERAGE: ").append(average).append("]");
                builder.append("[FORK JOIN DURATION: ").append(duration).append("]");
                builder.append("[FORK JOIN EFFICIENCY: ").append((double) singleDuration / duration).append("]");

                LOGGER.info(builder.toString());
            }
        }


        //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise1/AverageLengthForkJoin.java

        package org.LabExercises.Exercise1;

        import org.glassfish.grizzly.utils.Pair;

        import java.util.Arrays;
        import java.util.concurrent.ExecutionException;
        import java.util.concurrent.RecursiveTask;

        public class AverageLengthForkJoin extends RecursiveTask<Pair<Integer, Integer>> {
            private final String[][] tokens;
            private final int index;
```

```
        public AverageLengthForkJoin(int index, String[][] tokens) {
            this.tokens = tokens;
            this.index = index;
        }

        @Override
        protected Pair<Integer, Integer> compute() {
            if(index >= tokens.length) return new Pair<>(0, 0);

            var rhs = new AverageLengthForkJoin(index + 1, tokens);
            rhs.fork();

            try {
                var other = rhs.get();
                var total = Arrays.stream(tokens[index]).mapToInt(String::length).sum();
                other.setFirst(total + other.getFirst());
                other.setSecond(other.getSecond() + tokens[index].length);
                return other;
            } catch(InterruptedException | ExecutionException e) {
                throw new RuntimeException(e);
            }
        }
    }

    // ./Lab4/Lab4/src/main/java/org/LabExercises/Exercise1/TextAnalyzer.java

    package org.LabExercises.Exercise1;

    import opennlp.tools.lemmatizer.LemmatizerME;
```

```
import opennlp.tools.lemmatizer.LemmatizerModel;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import org.LabExercises.Exercise3.CommonWordsForkJoin;
import org.springframework.data.util.StreamUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.ForkJoinPool;
import java.util.stream.Stream;

public class TextAnalyzer {
    private static final String SENTENCE_MODEL_PATH =
"LanguageModels/en-sent.bin";
    private static final String TOKEN_MODEL_PATH =
"LanguageModels/en-token.bin";
    private static final String LEMMATIZER_MODEL_PATH =
"LanguageModels/en-lemmatizer.bin";
    private static final String POSTAGGER_MODEL_PATH =
"LanguageModels/en-pos-maxent.bin";
    private static final String NOT_VALID_SYMBOLS = "[^a-zA-Z\s\n]";
    private static final String LINE_FEED = "\n";
    private static final String EMPTY_SPACE = "";
```

```
        private final ForkJoinPool pool = ForkJoinPool.commonPool();
        private final SentenceDetectorME sentenceDetector;
        private final TokenizerME tokenizer;

        private static TextAnalyzer fileStringReader;

        public static TextAnalyzer getInstance() {
            if (fileStringReader == null) {
                fileStringReader = new TextAnalyzer();
            }
            return fileStringReader;
        }

        private TextAnalyzer() {
            try {
                var sentModelIn = new FileInputStream(SENTENCE_MODEL_PATH);
                var sentModel = new SentenceModel(sentModelIn);
                sentenceDetector = new SentenceDetectorME(sentModel);
                var tokModelIn = new FileInputStream(TOKEN_MODEL_PATH);
                var tokModel = new TokenizerModel(tokModelIn);
                tokenizer = new TokenizerME(tokModel);
            } catch(IOException e) {
                throw new RuntimeException(e);
            }
        }

        public Stream<String[]> getTokenStream(String fileName) {
            try {
                String text = readFile(fileName);
                return Arrays.stream(sentenceDetector.sentDetect(text))
```

```
                .map(s -> s.replaceAll(NOT_VALID_SYMBOLS, EMPTY_SPACE))
                .map(tokenizer::tokenize);
        } catch(FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    public double getAverageLength(String filePath) {
        var average = getTokenStream(filePath).flatMap(Arrays::stream)
            .mapToDouble(String::length).average();
        return average.isPresent() ? average.getAsDouble() : 0;
    }

    public double getAverageLengthForkJoin(String filePath) {
        var tokens = getTokenStream(filePath).toArray(String[][]::new);
        var res = pool.invoke(new AverageLengthForkJoin(0, tokens));
        return (double) res.getFirst() / res.getSecond();
    }

    public String readFile(String filePath) throws FileNotFoundException {
        var file = new File(filePath);
        var scanner = new Scanner(file);
        var builder = new StringBuilder();
        while(scanner.hasNextLine()) {
            builder.append(scanner.nextLine()).append(LINE_FEED);
        }
        return builder.toString();
    }
}
```

```
//
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise4/KeyWordsFinder.java

package org.LabExercises.Exercise4;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;
import java.util.concurrent.ForkJoinPool;

public class KeyWordsFinder {
    private final ForkJoinPool pool = ForkJoinPool.commonPool();

    public KeyWordsFinder() {}

    public static void main(String[] args) {
        var finder = new KeyWordsFinder();
    }

    public ArrayList<String> findDocuments(String rootDirectory,
List<String> keyWords) {
        var keyWordsLower =
keyWords.stream().map(String::toLowerCase).toList();
        var res = new ArrayList<String>();
        try {
            doFindDocuments(rootDirectory, keyWordsLower, res);
        } catch(FileNotFoundException e) {
            throw new RuntimeException(e);
        }
```

```
        return res;
    }

    private void doFindDocuments(String rootDirectory, List<String>
keyWords, ArrayList<String> res) throws FileNotFoundException {
        var keyWordsLower =
keyWords.stream().map(String::toLowerCase).toList();
        var dir = new File(rootDirectory);
        for(var file : Objects.requireNonNull(dir.listFiles())) {
            if(file.isDirectory()) {
                doFindDocuments(file.getAbsolutePath(),
keyWordsLower, res);
            } else {
                var scanner = new Scanner(file);
                while(scanner.hasNextLine()) {
                    var line = scanner.nextLine().toLowerCase();
                    if(keyWords.stream().anyMatch(line::contains)) {
                        res.add(file.getAbsolutePath());
                    }
                }
            }
        }
    }

    public ArrayList<String> findDocumentsForkJoin(String rootDirectory,
List<String> keyWords) {
        var keyWordsLower =
keyWords.stream().map(String::toLowerCase).toList();
        return pool.invoke(new FolderTask(rootDirectory,
keyWordsLower));
    }
}
```

```
// ./Lab4/Lab4/src/main/java/org/LabExercises/Exercise4/FileTask.java

package org.LabExercises.Exercise4;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.RecursiveTask;

public class FileTask extends RecursiveTask<Boolean> {
	private final List<String> keyWords;
	public final File file;

	public FileTask(File file, List<String> keyWords) {
		this.file = file;
		this.keyWords = keyWords;
	}

	@Override
	protected Boolean compute() {
		try {
			var scanner = new Scanner(file);
			while(scanner.hasNextLine()) {
				var line = scanner.nextLine().toLowerCase();
				if(keyWords.stream().anyMatch(line::contains)) {
					return true;
				}
			}
			return false;
```

```
            } catch(FileNotFoundException e) {
                throw new RuntimeException(e);
            }
        }
    }


    //
./Lab4/Lab4/src/main/java/org/LabExercises/Exercise4/KeyWordsFinderTester.java

    package org.LabExercises.Exercise4;

    import org.apache.logging.log4j.LogManager;
    import org.apache.logging.log4j.Logger;

    import java.util.List;

    public class KeyWordsFinderTester {
        private static final Logger LOGGER =
LogManager.getLogger("Exercise4");
        private static final String DIRECTORY = "ResourcesExercise4/Test";

        public static void main(String[] args) {
            testProfile();
        }

        private static void testProfile() {
            var keyWordsFinder = new KeyWordsFinder();
            var keyWords = List.of("c++", "java", "python");

            var builder = new StringBuilder();
```

```
                var singleStartTime = System.currentTimeMillis();
                keyWordsFinder.findDocuments(DIRECTORY, keyWords);
                var singleDuration = System.currentTimeMillis() - singleStartTime;

                builder.append("[SINGLE DURATION: ").append(singleDuration).append("]");

                var startTime = System.currentTimeMillis();
                keyWordsFinder.findDocumentsForkJoin(DIRECTORY, keyWords);
                var duration = System.currentTimeMillis() - startTime;

                builder.append("[FORK JOIN DURATION: ").append(duration).append("]");
                builder.append("[FORK JOIN EFFICIENCY: ").append((double) singleDuration / duration).append("]");

                LOGGER.info(builder.toString());
            }
        }


        // ./Lab4/Lab4/src/main/java/org/LabExercises/Exercise4/FolderTask.java

        package org.LabExercises.Exercise4;

        import java.io.File;
        import java.util.List;
        import java.util.Objects;
        import java.util.concurrent.RecursiveTask;
        import java.util.ArrayList;
```

```
public class FolderTask extends RecursiveTask<ArrayList<String>> {
    private final ArrayList<FolderTask> folderTasks = new ArrayList<>();
    private final ArrayList<FileTask> fileTasks = new ArrayList<>();
    private final List<String> keyWords;
    private final File dir;

    public FolderTask(String name, List<String> keyWords) {
        this.dir = new File(name);
        this.keyWords = keyWords;
    }

    @Override
    protected ArrayList<String> compute() {
        for(var file : Objects.requireNonNull(dir.listFiles())) {
            if(file.isDirectory()) {
                var rhs = new FolderTask(file.getAbsolutePath(),
keyWords);
                rhs.fork();
                folderTasks.add(rhs);
            } else {
                var rhs = new FileTask(file, keyWords);
                rhs.fork();
                fileTasks.add(rhs);
            }
        }
        var res = new ArrayList<String>();
        for(var fileTask : fileTasks) {
            if(fileTask.join()) {
                res.add(fileTask.file.getAbsolutePath());
            }
        }
        for(var folderTask : folderTasks) {
```

```
                res.addAll(folderTask.join());
            }
            return res;
        }
    }
```